Documentation for low-level Fed9USoftware (Fed9UVmeDevice and Fed9UVmeBase)

Matthew Pearson (Matthew.Pearson@cern.ch)

Gareth Rogers (ggr1@aber.ac.uk)

Contents

The Fed9U Namespace

The Front End Driver (FED) is a 9U 400mm VME64x card designed for reading out the CMS silicon tracker signals transmitted by the APV25 analogue pipeline ASICs. The signals are transmitted to each FED via 96 optical fibres at a total input rate corresponding to 3 Gbytes/s. The FED digitizes the signals and processes the data digitally by applying algorithms for pedestal and common mode noise subtraction. The FED home page can be found here [1].

The Fed9U software controls the FED registers and the data readout from the FED buffers. It is used to configure the run parameters, readout the events and provides format checking of events. It is also capable of monitoring the temperature of the hottest regions of the FED PCB and the voltages supplied to the FED components.

All Fed9U software is contained within the Fed9U namespace. It is structured into two layers, Fed9UUtils and Fed9UDevice, each with its own header and dynamically linked library. Fed9UUtils is a collection of utility classes that perform tasks that are used throughout the Fed9U namespace. Fed9UDevice contains the libraries and header file for accessing the FED hardware and is dependent upon Fed9UUtils. There is also a group of standalone executables that can be used to perform a variety of testing and debugging of a FED. They allow the FED to be installed as a stand alone piece of hardware, independent from any other component of the tracker readout chain.

The Fed9UVmeDevice Class

Overview

The aim of Fed9UVmeDevice is to provide a simple interface to the FED firmware. The firmware provides a set of methods that can be used to configure and control the FED. Fed9UVmeDevice accesses the FED through the class Fed9UVmeBase. This provides access to the FED registers that is similar in layout to the FED firmware. Fed9UVmeDevice provides an abstraction from this layout to an interface which is not constrained by the FED firmware structure.

The Abstract Base Class (ABC) Fed9UABC, provided by the Fed9UUtils library, is the base class from which Fed9UVmeDevice is derived. It provides the interface which is used by Fed9UVmeDevice for accessing the FED. The inherited interface gives the user access to the individual FED register at their lowest level of granularity. Fed9UVmeDevice extends on this interface to provide methods for the configuration and validation of the whole FED in a minimum amount of method calls.

Using Fed9UVmeDevice

A comprehensive list and description of all the methods provided by Fed9UVmeDevice class (as well as all the classes in the Fed9U namespace) are available on the FED savannah project site [2].

Instantiating a Fed9UVmeDevice Object

There is only one public Fed9UVmeDevice constructor provided; both the copy and assignment constructors are declared private and are not implemented. The constructor takes a single argument, a constant reference to a Fed9UDescription object, of which it makes a local copy. This Fed9UDescription class is also derived from Fed9UABC and it is used to store the FED settings in software. Fed9UVmeDevice is continually updating its local Fed9UDescription and so the User must update their copy in order to ensure it reflects the current FED values.

A default constructed Fed9UDescription contains a set of valid settings for the FED, which would allow the user to immediately start using a FED to take data in the default data acquisition (DAQ) mode. There are two settings that are unique to a FED which the user must ensure are configured correctly before instantiating the Fed9UVmeDevice object. These are the slot number and the location of the Fed9UHalAddress table. The slot number represents the slot in the crate that contains the relevant FED. The Fed9UHalAddress table contains the addresses of the FED registers and is required if Fed9UVmeBase is to function correctly, see the Fed9UVmeBase section of this document for more details. If the PC is controlling multiple FED crates then the crate number must be correctly configured; however in a single crate environment the default value is sufficient.

Setting up a FED

A FED can be used to readout data with a Fed9UVmeDevice object constructed with a default constructed description. It is recommended that the user configure the description object first to ensure that the FED runs as desired.

The simplest way to configure a FED using a Fed9UVmeDevice object is with the initialisation methods provided by Fed9UVmeDevice. They can be used to setup the whole FED using a few method calls. Individual settings on the FED can be modified using the other Fed9UVmeDevice methods provided to access the individual registers.

There are three different initialisation methods provided for configuring the FED: *init*, *initStrips* and *initAuxiliaryChips*. *init* is called when there are major changes required to the run parameters, or a new Fed9UDescription object is loaded. It will setup all the FPGA parameters and the TTCrx device. The *initStrips* is only needed in processed raw or zero suppressed data acquisition modes. It sets up the strip pedestals, cluster thresholds and the valid strip settings. *initAuxiliaryChips* sets up the temperature and voltage monitoring parameters.

In the event that the FED is incorrectly configured it is possible to return the FED settings to their hardware default values by performing a FED reset. It will clear all the FED registers, apart from the TTCrx chip, the temperature monitors, pedestals, disabled strips, high and low cluster thresholds and the clock source. There are also FE and BE FPGA soft resets which clear the FPGA logic, but do not affect any of the register settings. The TTCrx, temperature monitors and voltage monitor all have their own hard resets which clear all settings. The clock source is unaffected by resets.

Data Readout

There are two different ways a FED can respond to incoming data. It can either detect incoming triggers or detect incoming frames. These are called scope mode and frame finding mode respectively. When in scope mode no data processing is performed and both the APV tick mark and the frame are present in the event data. When in frame finding mode only the frame containing the strip data is present in the event data. There are three different frame finding modes: virgin raw data mode, where no data processing is performed; processed raw data mode, where only strip reordering and pedestal subtraction is performed; and zero suppression mode, where strip reordering, pedestal subtraction, common mode median subtraction and cluster finding is performed. Scope mode, virgin raw, processed raw and zero suppressed data modes make up the four data acquisition (DAQ) modes available on the FED.

Once the FED has been properly configured in the desired DAQ mode event readout can be performed. When in scope mode the FED will respond to triggers rather than frames and no optical input is required on the FE FPGAs. In the frame finding modes the FE FPGAs must be driven.

When reading via VME, unread events will be stored in the QDR memory until it has been told the VME event buffer is empty. At this point the next event is sent from the QDR to the VME event buffer. An event can be checked for using the method hasEvent(), which has three different return values: no event present, event present, and last fragment of an event. A user will typically only be concerned whether or not an event is present, in which case the return can be treated as boolean, where a false represents no event and a true represents an event. If there is an event present the method getCompleteEvent should be called. This must be provided with a suitably large buffer to store the event. It gives the user the option of using a block transfer if their hardware supports it. Once read the data should be passed to the Fed9UEvent class, which can check the event format for errors. After getCompleteEvent has completed the FED will prepare the next event to be readout. The number of events that have been received and are left to be readout can be monitored using the method getBeEventCounterStatus.

Monitoring the FED

The FED contains two hardware monitoring devices. There are temperature monitors present on each FPGA and a voltage monitoring device. Each temperature sensor can monitor the FPGA temperature and the temperature of the sensor itself. Each temperature can have an independent maximum temperature limit, which when exceeded will be flagged in a status register on the device. There is also a critical temperature limit, which, if exceeded, causes power to be cut to the FED to prevent damage to the components. This is a last resort as the FED will have to be switched back on at the physical FED location, and cannot be performed via software.

The voltage monitoring device which 2.5V, 3.3V, 5V, 12V, its core voltage and the supply voltage. Each of these can have lower and upper limits set. If the voltage is outside a limit then it will be flagged in a status register on the device. The temperature also has upper and lower limits, which will be flagged if the temperature goes out of bounds. It is possible for the voltage monitor to measure an external temperature as well as the internal chip temperature, however in the current FED design this ability is not used.

Using the EPROM

The EPROM is a 2 kilobyte area of non-volatile memory that is split into four equal sized sections. It is possible to write protect one quarter, half or all of the EPROM memory. In the final system it is expected that the write protection level will be locked in the hardware giving the user access to only certain quadrants of memory. Currently, however, a user must set the write only level after each power up, otherwise the EPROM memory cannot be accessed. Due to the current nature of the write only level and the lack of any definition of what the EPROM memory should be used for the user should be careful that they do not overwrite any important data that someone else may have written; and also care should be taken not to write over the FED serial number, for which a specific set/get method exists in Fed9UVmeDevice. At present the only bounds checking that takes place is that the user is writing to a valid area of memory (i.e. inside the 2KB region) -- there is no software warning if a user attempts to write to a write protected area. If a write protected area is written the write will appear to have succeeded, and there is no warning from either the hardware or software. The only mechanism that can be used to check is by reading back from that area of memory to ensure the written data is now present.

Fed9UVmeDeviceException

ICUtils::ICException provides a set of pre-processor macros that are used to create the framework for an exception class derived from ICUtils::ICException, which can then be customised to the meet the exception class requirements. It is not provided as part of the Fed9U namespace, but is provided as a general utility in the ICUtils namespace.

Fed9UVmeDeviceException is an exception class used to describe the various error conditions that can occur during the use of Fed9UVmeDevice. It is derived from ICUtils::ICException, which is derived from std::exception. It contains an exception code list, which can be used to identify the type of error thrown. It also contains an error string that identifies where the error was thrown from and gives some information on the error conditions that generated the exception.

It is expected that Fed9UVmeDevice will only throw exceptions of type Fed9UVmeDeviceException, although it is possible that an included class may throw an exception of some other type. Fed9UVmeDevice is capable of catching ICUtils::ICExceptions, which is what included classes are expected to throw (or classes derived from this). It is advised that any classes including Fed9UVmeDevice are prepared to handle other exception types.

There are basically two different errors that will be thrown from Fed9UVmeDevice. The first is that which has been re-thrown from an included class and the other is an out of bounds error from one of the methods setting a new value to the FED. In the case of a re-thrown error the documentation for that class should be consulted. A common type of error encountered in this case may declare something like "cannot write to address 010004 with 4 bytes of data". This typically means that the base address has been incorrectly set and there is no FED in that slot.

The Fed9UVmeBase Package

Introduction

The Fed9UVmeBase class contains methods which enable FED configuration and data readout. It sits below Fed9UVmeDevice and acts as an interface between the user friendly methods in Fed9UVmeDevice and the FED firmware commands. It contains methods to mirror the firmware in the FED FPGA chips. Therefore, for every firmware command there is a corresponding method in the class. This class is a low level part of the Fed9USoftware and should not be used directly by a user.

For a detailed documentation on the FED commands see the FED FPGA documents (http://www.te.rl.ac.uk/esdg/cms-fed/qa_web/documentation/). These documents contain a description of each command and details such as the command format, timing issues, etc.

Low level utility classes

The Fed9UVmeBase class uses two additional utility classes to help construct the command and send it to the FED using the Hardware Access Library (HAL). They are called **Fed9UConstructCommand** and **Fed9UHalInterface**. Fed9UConstructCommand is used by every method in Fed9UVmeBase to construct the bit pattern that is sent to the FED. It uses command specific information and user defined information (for example, the data to write to the FED). Fed9UHalInterface is used by every method in Fed9UVmeBase to provide an interface to HAL to allow writes/reads to/from the FED VME space. It requires a HAL VME address table as input to its constructor, in order to initialize the VME interface in HAL.

Fed9UVmeBase

The methods in Fed9UVmeBase address three types of FPGA on the FED: the 8 front end FPGAs, the back end FPGA and the VME interface FPGA. Every front end FPGA method requires an input telling it which FPGA to address (1-8 are the front end FPGAs, and 15 is used to address all front end FPGAs). These numbers mirror the FED firmware FPGA numbering scheme. Internally other FPGA numbers are used to address the back end FPGA (10 and 16), and the software trigger (9). To address the VME FPGA the software uses the VME memory map defined in the HAL address table.

Constructor

The location of the HAL address table text file must be specified as a parameter to the Fed9UVmeBase constructor, along with the FED VME base address (in base 16) and crate number (the crate number in effect selects the PCI-VME interface, of which there are one per crate).

```
Fed9UVmeBase::Fed9UVmeBase(unsigned long baseAddress,
                           string theAddressTable,
                           unsigned short crateNumber)
```

One could use the Fed9UVmeBase class independently of the rest of the Fed9USoftware, in order to facilitate a low level FED test, by the simple constructor call above and a set of subsequent method calls (described below). However, using it via Fed9UVmeDevice is advised and is much more user friendly.

Read/Write commands

The read/write methods take as input a read/write flag and a vector holding the data to write. Every method is also passed a reference which can be used to hold the data when reading from the FED. Each method can throw an exception in the form of a Fed9UVmeBaseException object. For front end FPGA commands the FPGA number (1-8 or 15) is a required input. An example of a read/write front end FPGA method is the "load threshold" command (which loads the threshold for frame finding in the front end):

```
void Fed9UVmeBase::feCommandLoadThresh(unsigned short selectedFpga,
                                       bool read,
                                       const vector<unsigned long> &arguments,
                                       vector<unsigned long> &readArguments)
                                       throw (Fed9UVmeBaseException)
```

In the Fed9UVmeBase source code this method will be accompanied by a comment explaining what the command does, and what each input parameter means. There is a common naming convention throughout Fed9UVmeBase, so that *selectedFpga* always means the FPGA number to address (1-8 or 15), *read* is always the read/write flag, the *arguments* vector always holds the data to write (this can be a dummy zero length vector when reading) and the *readArguments* vector always holds the data read from the FED when reading (this can be a dummy zero length vector when writing).

Read only commands

The read only commands have a simpler interface. For example, the method which reads the front end FPGA firmware IDs is:

```
void Fed9UVmeBase::feCommandFirmwareIdReg(unsigned short selectedFpga,
                                          unsigned long &readArguments)
                                          throw (Fed9UVmeBaseException)
```

and the method to read the back end FPGA status register is simply:

```
void Fed9UVmeBase::beCommandBeStatusRegister(unsigned long&readArguments)
                                             throw (Fed9UVmeBaseException)
```

Additional commands and private methods

In addition to the basic format outlined above there are a few commands that have a slightly more complicated interface. One example is the control command for the LM82 temperature sensor on the back end FPGA. As well as the standard read/write interface presented above there are additional I^2C parameters that return any I^2C error or busy flags. There are comments in the source code as and when these situations arise.

There are also numerous private methods that factor out common code from many of the public methods. These perform tasks such as error checking, data formatting/unformatting, using the Fed9UHalInterface and Fed9UConstructCommand classes and a few FED commands factored out of the public commands.

Summary

Fed9UVmeDevice is a derived class of Fed9UABC and provides access to the FED registers. It has been designed to allow the user a simple and easy to use interface that is flexible enough to meet all users needs. It relies on a variety of support classes that range from exception handling to classes that correspond directly to devices on the FED. It has been integrated into the Fed9USoftware and serves as a base class for Fed9UDevice, which provides additional higher level methods.

Operating below Fed9UVmeDevice is Fed9UVmeBase, which provides an interface for Fed9UVmeDevice to the actual FED commands in firmware. Fed9UVmeDevice acts as a user friendly interface to Fed9UVmeBase.

References

[1]FED UK - http://www.te.rl.ac.uk/esdg/cms-fed/qa_web/

[2]Fed9U savannah project page - http://savannah.cern.ch/projects/fed9usoftware/

[3]Authors email address - ggr1@aber.ac.uk